ARL-TN-1112 ● Mar 2022

Running Virtual Environment-Based Experiments Online

by Benjamin T Files, Kimberly A Pollard, Ashley H Oiknine, and Bianca Dalangin

ARL-TN-1112 ● MAR 2022

Running Virtual Environment-Based Experiments Online

Benjamin T Files and Kimberly A Pollard
DEVCOM Army Research Laboratory

Ashley H Oiknine and Bianca Dalangin
DCS Corporation

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* March 2022	**2. REPORT TYPE** Technical Note	**3. DATES COVERED (From - To)** 1 October 2020–1 February 2022
4. TITLE AND SUBTITLE Running Virtual Environment-Based Experiments Online		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Benjamin T Files, Kimberly A Pollard, Ashley H Oiknine, and Bianca Dalangin		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLH-FA Playa Vista, CA		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-1112
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID IDs: Benjamin T Files, 0000-0002-1141-7886; Kimberly A Pollard, 0000-0002-5849-1987; Ashley H Oiknine, 0000-0001-9092-7958; Bianca Dalangin, 0000-0002-6084-2803

14. ABSTRACT
Conducting behavioral experiments online enables researchers to obtain large sample sizes quickly. Existing tools for online experiments are primarily suited for text and 2-D images, while bringing 3-D virtual environment experiences to the web for research remains a challenge. This technical note describes a process to present complex 3-D virtual environments to online participants and collect the resulting data. The process starts with a virtual environment built in Unity and describes the steps to display the virtual environment in the context of a Qualtrics questionnaire, including how to save data into a Qualtrics user record. Code and examples are provided with the hope that other researchers can conveniently follow this process to conduct online behavioral experiments using 3-D virtual environments.

15. SUBJECT TERMS
virtual environments, online experiments, behavioral sciences, online methods, remote testing, Unity, Qualtrics

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Benjamin T Files
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	22	**19b. TELEPHONE NUMBER (Include area code)** (310) 577-6212

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction and Overview

Conducting behavioral experiments online is convenient and effective, but running behavioral experiments online can be technically challenging.[1] Excellent resources are available for running behavioral experiments online with tools focused on text and/or 2-D graphics (e.g., psychopy[2] and jsPsych[3]). For 3-D virtual environment-based research, additional tools are needed.

This note describes one approach to putting a virtual environment-based behavioral experiment online using a combination of Unity, GitHub Pages, and Qualtrics (Fig. 1). Alternative technologies are available to implement virtual environments on the web. For example, the Unreal Engine is capable of building experiences for the web and could be used in place of Unity. For this note, we focus on Unity, GitHub Pages, and Qualtrics as technologies that are relatively accessible, intuitive, and widely available to researchers. This note does not describe the process of generating a virtual environment with Unity* or provide comprehensive instructions for building questionnaires in Qualtrics†. Instead, the focus is on getting these technologies to work together to enable a researcher to run their experiment in participants' browsers and record results without requiring additional steps from the participant.

This note will first describe how to put a virtual environment-based behavioral experiment online, then how to save the data from the experiment. Towards the end of this note, a more concise step-by-step guide is provided.

The following section will describe the three technologies we use for running online virtual environment-based behavioral experiments.

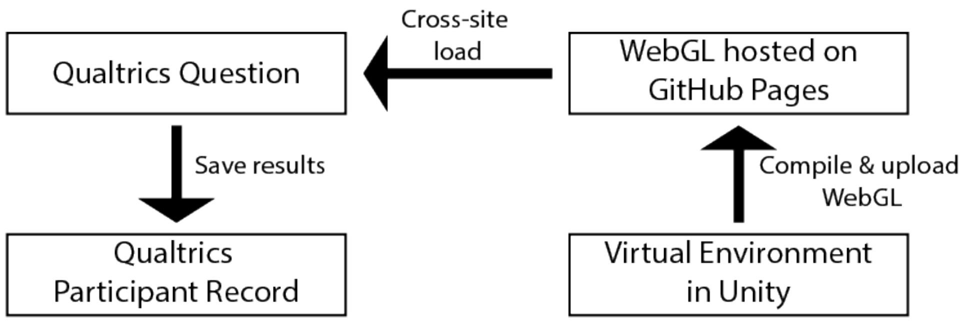

Qualtrics Question
Cross-site load
WebGL hosted on GitHub Pages
Save results
Compile & upload WebGL
Qualtrics Participant Record
Virtual Environment in Unity

Fig. 1 Flowchart with a high-level representation of how Unity, GitHub Pages, and Qualtrics work together to enable online virtual environment-based experiments

*Tutorials and other learning materials are available on the Unity learning website.

†The Qualtrics documentation website provides detailed instructions for building questionnaires.

1.1 Unity

Unity is a cross-platform engine for games and other virtual/interactive experiences. It is also a tool for building these experiences. It includes a graphical interface for building environments as well as a scripting interface for code to specify behaviors and influence the flow of the game or experience. Although it is relatively easy to build and run simple games and experiences on Unity, making a virtual environment and scripting it for the purposes of a behavioral experiment on this platform requires some expertise. This note will not attempt to provide guidance on building a Unity-based behavioral experiment per se, and instead focuses on taking an existing virtual experiment and making it available over the web.

Unity is commercial software. The cost of using Unity depends on the details of who is using it and for what purpose. Unity provides free licenses for personal/small organization use as well as for student use.

1.2 GitHub Pages

GitHub Pages is a web hosting service integrated with GitHub. GitHub is an online development platform that integrates with Git, a source control tool. Git and GitHub are both powerful and complex tools, but this note will just focus on GitHub Pages as a convenient web host for some of the components of the online behavioral experiment.

GitHub Pages can be used for free, although it is available as a paid service as well. The primary difference is that free accounts may not make their repositories private. This means that anyone may view the code and data posted to GitHub Pages (i.e., the repository is *public*). This limitation is not relevant to our use-case, because we need the behavioral experiment to be available to participants, so it needs to be public.

1.3 Qualtrics

Qualtrics is a web-based platform that can create and deploy questionnaires with sophisticated structures and interactive capabilities. Qualtrics has various built-in question templates to help create a questionnaire from standard question types such as multiple choice and free-response. In addition, Qualtrics allows creators to add HTML to questions to create custom question types. Qualtrics also allows custom functionality by applying JavaScript code to questions. Qualtrics provides a JavaScript API to interact with Qualtrics-specific functionality, including storing custom data within the questionnaire response record. These features provide the flexibility to include a WebGL-based virtual environment within a Qualtrics question.

Qualtrics is not a free service, although Qualtrics offers demonstration accounts for free. A paid subscription is required to use some of the features leveraged in the process described in this note.

Having reviewed these three technologies, the remainder of this note is organized as follows: First it covers how to display the Unity experience within Qualtrics, and then it covers how to pipe data from the Unity experience into Qualtrics.

2. Showing a Virtual Environment as a Qualtrics Question

At a high level, the process of showing a Virtual Environment as a Qualtrics question involves first using Unity to build the project as WebGL files, then putting those WebGL files on a web server, and lastly, loading the WebGL into the Qualtrics question. Here, we provide some brief background on WebGL and then provide details of each of these steps.

2.1 Background: WebGL

For a user to interact with a Unity project without having access to Unity, a build must be created for the target operating system or platform. Unity can build projects targeting WebGL, instead of a particular operating system like Windows. WebGL uses HTML5 to render 3-D content within a web page. JavaScript (a scripting language that works in most web browsers) then enables features like animation and interactivity. Many browsers support WebGL, but it is disabled or prohibited in some cases. The WebGL web page has a quick test that shows whether WebGL is enabled and documentation for enabling it in some situations.

2.2 Unity's WebGL Build

When a Unity project builds with WebGL as the target, Unity generates a folder with an index.html file and two subfolders, *Build* and *TemplateData*. The Build subfolder has the JavaScript and data for the WebGL. TemplateData has some images supporting the WebGL. For this note, the TemplateData folder includes images to support the loading bar and so on, and importantly it has the *style.css* file. This controls the look and positioning of the WebGL canvas, so it is important for functionality.

Some functionality that is available when building a Unity project for a non-WebGL target is prohibited or needs special handling in a WebGL build. Some examples include locking or hiding the mouse cursor, accessing network resources, accessing the local file system, and playing audio. These special cases will not be covered in this note, but they are explained in the Unity documentation.

The following is a listing of files and folders generated by Unity's build process for an example build called *qualifier*.

```
qualifier
│   index.html
│
├───Build
│       qualifier.data.unityweb
│       qualifier.framework.js.unityweb
│       qualifier.loader.js
│       qualifier.wasm.unityweb
│
└───TemplateData
        favicon.ico
        fullscreen-button.png
        progress-bar-empty-dark.png
        progress-bar-empty-light.png
        progress-bar-full-dark.png
        progress-bar-full-light.png
        style.css
        unity-logo-dark.png
        unity-logo-light.png
        webgl-logo.png
```

The Unity *project settings* dialog has some compression options that could be used to speed load the virtual environment. Some compression options require configuration of the web host to send special headers; configuring that goes beyond the scope of this document. Our recommendation is to use the uncompressed option unless load times become unacceptable.

The WebGL build includes a simple web page, index.html, that can be opened with a web browser. This shows a loading bar as it attempts to load the WebGL experience, but file permission errors will usually prevent the loading from succeeding. For successful loading, the page needs to be served by a local web server[*], which can be done automatically using Unity's *Build & Run* option. This will allow the experience to work in the browser on the local machine, but it will not let anyone else see it. That works well for testing, but to deploy the experience to Qualtrics, this WebGL build needs to be somewhere on the internet so others can see it.

2.3 Hosting WebGL on the Web

Any web server should work for hosting the WebGL build on the Internet. The web server needs to be configured to allow cross-origin requests of WebGL textures[†]. One option is a GitHub Pages account (see https://pages.github.com/ for details on setting up an account and configuring a GitHub repository with pages). It is free and easy to use, and it does not require any special configuration to host a WebGL virtual environment for use in Qualtrics. Using Git, you can push the WebGL build folder (generated by Unity, see above) to a GitHub project configured to use GitHub Pages. Although there is sometimes a delay of a few minutes, you can confirm the WebGL build worked by pointing a browser at the index.html on GitHub Pages.

Once the code and data that drive the WebGL experience are on the internet for anybody to see (i.e., set to public), the behavioral experiment should run. This is fine for demonstration and testing, but it will not save any data or results. Next we turn to how to embed this experience in a Qualtrics question so that the results can be saved.

2.4 Qualtrics Question and HTML

With the WebGL experience hosted on the web, we turn to presenting the experience in the context of a Qualtrics question. This requires a few steps. First we create a Qualtrics question within a questionnaire. The "Text/Graphic" question type creates a blank template, which we can edit using the HTML view (Fig. 2).

[*]Running a local web server is not as difficult as it might sound. For example, Python provides a one-line command that starts up a simple local web server.
[†]See Using textures in WebGL - Web APIs | MDN for information about cross-origin requests of WebGL textures.

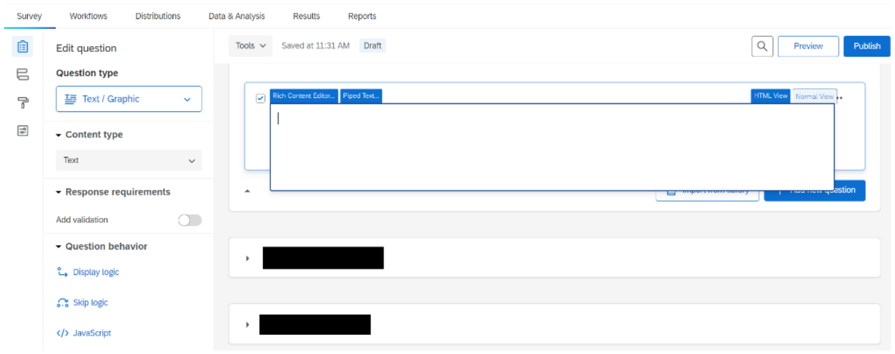


Fig. 2 Screen capture of the Qualtrics question editor with a new text/graphic question in HTML view

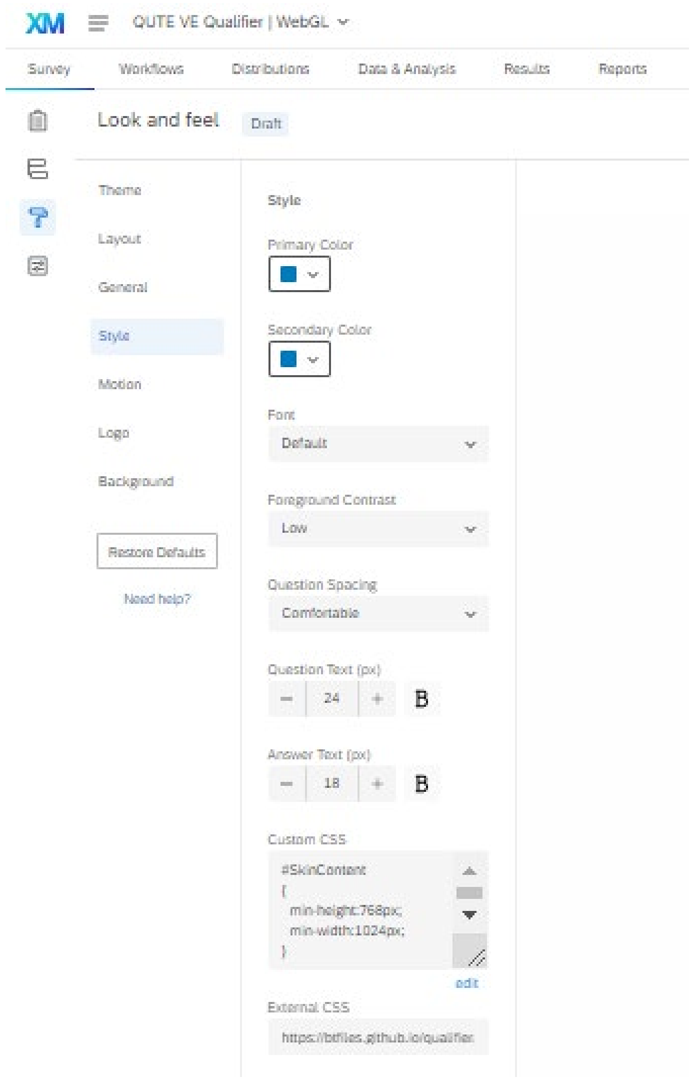
The following snippet of HTML is copied from the index.html (with some modifications explained below) that Unity generated when it built the WebGL build. The only piece of this that is not generic is the text "ExampleTitle", which may be changed to suit the project. The text is displayed to the participant near the bottom of the question. The other change is to set the fullscreen button div to be hidden (`visibility="hidden"`), because for this project we wanted to disable the fullscreen button. Here is text to paste into the Qualtrics HTML question:

```
<div id="unity-container" class="unity-desktop">
  <canvas id="unity-canvas"></canvas>
  <div id="unity-loading-bar">
    <div id="unity-logo">&nbsp;</div>
    <div id="unity-progress-bar-empty">
      <div id="unity-progress-bar-full">&nbsp;</div>
    </div>
  </div>
  <div id="unity-footer">
    <div id="unity-webgl-logo">&nbsp;</div>
    <div visibility="hidden" id="unity-fullscreen-button">&nbsp;</div>
    <div id="unity-build-title">ExampleTitle</div>
  </div>
</div>
```

Note: Much of this could be customized to obtain a different look. For example, the Unity footer could be omitted. That would require some changes to other pieces that reference the Unity footer, and such changes will not be covered here.

2.5 Qualtrics Style Sheet

For everything to look the same within the Qualtrics question as it does when testing locally or on the web, Qualtrics will need to use the Unity-generated cascading style sheet (CSS). To set up the Qualtrics questionnaire to use this CSS, select any question within the project and select the Look and Feel setting (a paint-roller icon on the left) and then click Style (Fig. 3).



Style

Primary Color

Secondary Color

Font: Default

Foreground Contrast: Low

Question Spacing: Comfortable

Question Text (px): 24

Answer Text (px): 18

Custom CSS

```
#SkinContent
{
  min-height:768px;
  min-width:1024px;
}
```

External CSS: https://btfiles.github.io/qualifier

Fig. 3 Screen capture of the Qualtrics style menu

Scrolling down, insert the URL for the style sheet into the external CSS field: `https://btfiles.github.io/qualifier/TemplateData/style.css` (In this example style sheet URL, "btfiles" is the GitHub Pages account name, "qualifier" is the name of the build folder, and "TemplateData/style.css" are the folder and filename Unity uses by default). This external style sheet handles the look and feel of the text and is important for the functionality of the loading bar.

After pasting the URL for the CSS file, insert code to designate the desired size of the question container, for example:

```
#SkinContent
{
  min-width:1024px;
  min-height:768px;
}
```

That custom CSS makes sure the question container is the right size for the WebGL canvas. We use 1024 × 768 pixels for the question size because we designed our virtual environment for that size. To use a different size, all instances of 1024 and 768 would need to be swapped for the new width and height, respectively.

2.6 Qualtrics JavaScript

Next, we need to add JavaScript to the question. Pressing the "add javascript" button on the question brings up a JavaScript template. Importantly, we need to put all the JavaScript that loads the WebGL into the "onload" section:

```
Qualtrics.SurveyEngine.addOnload(function()
{
    /*Place your JavaScript here to run when the page loads*/
    this.hideNextButton();
    this.hidePreviousButton();

    var q = this;

    window.dataFun = function (str) {
        str = str.replace(/(?:\r\n|\r|\n)/g, ' <br> ');
        console.log("DataFun Says: " + str);
        Qualtrics.SurveyEngine.setEmbeddedData( 'CSVString', str );
        q.showNextButton();
        console.log("calling clickNextButton.");
        q.clickNextButton();
    };

    var buildUrl = "https://btfiles.github.io/qualifier/Build";
    var loaderUrl = buildUrl + "/qualifier.loader.js";

    jQuery.getScript(loaderUrl, function (data, textStatus, jqhxr) {
        var config = {
            dataUrl: buildUrl + "/qualifier.data.unityweb",
            frameworkUrl: buildUrl + "/qualifier.framework.js.unityweb",
            codeUrl: buildUrl + "/qualifier.wasm.unityweb",
            streamingAssetsUrl: "StreamingAssets",
            companyName: "CCDC ARL",
            productName: "QUVE_Qualifier",
            productVersion: "0.1",
        };

        var container = document.querySelector("#unity-container");
        var canvas = document.querySelector("#unity-canvas");
        var loadingBar = document.querySelector("#unity-loading-bar");
        var progressBarFull = document.querySelector("#unity-progress-bar-full");
        var fullscreenButton = document.querySelector("#unity-fullscreen-button");
        fullscreenButton.style.display="none";

        if (/iPhone|iPad|iPod|Android/i.test(navigator.userAgent)) {
            container.className = "unity-mobile";
            config.devicePixelRatio = 1;
        } else {
            canvas.style.width = "1024px";
            canvas.style.height = "768px";
        }
        loadingBar.style.display = "block";

        createUnityInstance(canvas, config, (progress) => {
            progressBarFull.style.width = 100 * progress + "%";
        }).then((unityInstance) => {
            loadingBar.style.display = "none";
            fullscreenButton.onclick = () => {
                //unityInstance.SetFullscreen(1);
            };
            fullscreenButton.style.display="none";
        }).catch((message) => {
            alert(message);
        });
    });

});

Qualtrics.SurveyEngine.addOnReady(function()
{
    /*Place your JavaScript here to run when the page is fully displayed*/

});

Qualtrics.SurveyEngine.addOnUnload(function()
{
    /*Place your JavaScript here to run when the page is unloaded*/
    this.showNextButton();
    this.showPreviousButton();

});
```

Most of this JavaScript code is pasted from the Unity-generated index.html, but some adjustments allow it to work within Qualtrics.

In the Unity-generated index.html, there's a script (inside a `<script>` tag) that builds a script object, sets that object's onload property to that createUnityInstance function call, and then appends the script object to the document. This procedure is a best-practice for fast page loads when using JavaScript, but we need to put the code into a special onload context within Qualtrics.

Note: Everything above the buildUrl line is added to support data saving and interactions with Qualtrics; that will be covered in the next section.

The buildUrl value needs to be edited to reflect the actual URL of the build folder on the web. If the build name changes, then that line needs to change, as do all the places where the word “qualifier” appears (Unity names all the files after whatever the build name is). Similarly, if you change compression options in the Unity project settings, you might also need to change the extensions on some of those files.

The jQuery call loads the loader script that lives in the Build folder. This is necessary, because createUnityInstance is defined in the loader script. That call loads the script over the web and then calls the function defined in the second argument once the load is complete.

From there, the rest is copied from the index.html. However, the following lines disable and hide the “go fullscreen” button:

```
fullscreenButton.onclick = () => {
    //unityInstance.SetFullscreen(1);
};
fullscreenButton.style.display="none";
```

If the fullscreen button is desired for a project, then those lines should be pasted from the original WebGL build folder without modification.

Another addition is the calls to the Qualtrics Question API to hide any next/prev buttons so they are not drawn on top of the WebGL or accidentally pressed prematurely. Note these are unhidden down in the onUnload section.

3. Saving Data

We now have a Qualtrics question that loads our WebGL from GitHub Pages and presents the WebGL experience. If you were to run this now, you would run through to the end of your 3-D experience and then nothing would happen. This is progress, but for research, we also want to save data generated when the participant interacts with the Unity experience.

The experiment’s 3-D task will be collecting some data, as designed within the Unity project. We need to get that data out of the WebGL and into a location where it can be saved. An effective solution is to save it in a Qualtrics embedded field, found in the Qualtrics survey flow, so that the data become part of the participant’s record in Qualtrics. Fortunately, Unity allows us to write functions in JavaScript. Those functions have access to the global workspace (i.e., the JavaScript context of the web page, in this case the Qualtrics web page showing our question). Qualtrics gives us a way to use JavaScript to write to embedded fields. Rather than put Qualtrics-specific JavaScript inside the WebGL, we ask the WebGL to call a generic JavaScript function, and then we define that function in the Qualtrics question. This function serves as an intermediary between WebGL and Qualtrics.

As a result, data will flow from WebGL to JavaScript to Qualtrics. Once the data are written, we need to re-enable the next and previous Qualtrics buttons, and to help the participant out, we click the next button for them to advance past this question. The sections that follow provide more details.

3.1 Writing Data from WebGL to the JavaScript Intermediary

Within the Unity project, we need to designate that the project will need to use some JavaScript. This web page has the basics for making that happen: https://docs.unity3d.com/Manual/webgl-interactingwithbrowserscripting.html.

Using those instructions, Unity can then call a JavaScript function instead of attempting to write data to the local disk. This function will need to be defined in the global `window` workspace and called `dataFun(str)`. That function is not defined in the Unity project (and Unity can check if the function exists so things fail gracefully).

Following the Unity documentation recommendation, we put a .jslib file in the assets/plugins directory in the Unity project. It looks exactly like the example in the documents, but with the following function added:

```
LogStringJS: function (str) {
    var data_str = Pointer_stringify(str);
    try {
        window.dataFun(data_str);
    } catch (err) {
        console.log(err); // Error: "dataFun is not defined"
    }
    console.log(data_str);
},
```

Note the use of `Pointer_stringify`, which Unity defines and turns a Unity pointer into string data that JavaScript can actually use. Then, within the C# code for the Unity project, that function is imported as follows:

```
[DllImport("__Internal")]
    private static extern void LogStringJS(string str);

    ...

    public void SendDataToJS(string str)
    {
        LogStringJS(str);
    }
```

If the experiment were to run on a local computer, then the Unity project might have a call to save a data output file to disk for later analysis. Here, that call would be replaced with a call to SendDataToJS, which sets up the Unity project to send data to our JavaScript intermediary. Note that if you build and run to WebGL now, window.dataFun will not be defined, and you will see an error in the web console (along with whatever text data you were trying to save).

3.2 From the Intermediary to Qualtrics

Now we just need to define that window.dataFun function to put the data into a Qualtrics embedded field. The dataFun function is defined in the JavaScript on Qualtrics (it appears in the full JavaScipt above, but here it is as a reminder):

```
    window.dataFun = function (str) {
        str = str.replace(/(?:\r\n|\r|\n)/g, ' <br> ');
        console.log("DataFun Says: " + str);
        Qualtrics.SurveyEngine.setEmbeddedData( 'CSVString', str );
        q.showNextButton();
        console.log("calling clickNextButton.");
        q.clickNextButton();
    };
```

This function does a bit of processing (as well as printing to the console log for debugging).

It strips out newlines and replaces them with `<br>` and puts the data into a Qualtrics embedded data field. Embedded data cannot have line breaks, so that is why they are replaced with `<br>` tags. Once that processing is complete, the call to the Qualtrics API writes the string data to an embedded data field. In this example the field is called *CSVString*; if you use something different, you will want to update this code.

Embedded data fields are created within the Qualtrics' "survey flow" to allow JavaScript to write to them. The field should be created and named, but no value should be set for it. Embedded fields are limited to 20 KB, so if your embedded data are longer than that, you might need to break the data over multiple fields. Instructions for working with embedded fields can be found at https://www.qualtrics.com/support/survey-platform/survey-module/survey-flow/standard-elements/embedded-data/.

For simplicity, we are assuming that it makes sense to save a single result string at the end of the experiment. It would be possible to save several results throughout the course of the experiment. This would require some additional logic to write each result to a different Qualtrics embedded field.

Finally, this function reveals the next button and then clicks the next button. Note that `q` object is defined farther upscript as `this`, which provides access to the Qualtrics JS API.

The unload code still runs, so the next and previous buttons are re-enabled there.

4. Recipe

In summary, these are the steps and recommended order for getting a virtual environment-based experiment working within Qualtrics:

1) Create your virtual environment-based experiment in Unity.
2) Define and use LogStringJS within your Unity project to output data.
3) Make a build of your virtual environment-based experiment, targeting WebGL.
4) Push the build folder to your GitHub Pages project.
5) Create your Qualtrics questionnaire and question:
    1) Create a Text/Graphic question.
    2) Edit the question in HTML view and paste the HTML snippet.
    3) Press the Add JavaScript button and paste the JavaScript snippet, making edits to reflect your build name and GitHub Pages URL.
6) Set up the Qualtrics style:
    1) Go to the Look and Feel>Style button and enter the custom CSS snippet.
    2) Enter the GitHub Pages URL for the Unity-generated CSS file.
7) Create your embedded data fields in the Qualtrics survey flow.

5. Conclusion

This note has reviewed how to put a virtual environment-based behavioral experiment online by using Unity to build the project to WebGL, posting the WebGL on a web server, and then importing that WebGL experience into a Qualtrics question. It has also covered the mechanics of passing data from within the WebGL to the participant record within Qualtrics. We hope that this approach is useful to others who are interested in conducting virtual environment-based behavioral experiments.

6. References

1. Grootswagers T. A primer on running human behavioural experiments online. Behav Res. 2020;52:2283–2286. https://doi.org/10.3758/s13428-020-01395-3.

2. Peirce JW. PsychoPy—psychophysics software in Python. J Neuroscience Methods. 2007;162(1-2):8–13. https://doi.org/10.1016/j.jneumeth.2006.11.017.

3. de Leeuw JR. jsPsych: A JavaScript library for creating behavioral experiments in a web browser. Behav Res. 2015;47:1–12. https://doi.org/10.3758/s13428-014-0458-y.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
1 (PDF)	DEVCOM ARL FCDD RLH B T DAVIS BLDG 5400 RM C242 REDSTONE ARSENAL AL 35898-7290
1 (PDF)	DEVCOM ARL FCDD HSI J THOMAS 6662 GUNNER CIRCLE ABERDEEN PROVING GROUND MD 21005-5201
1 (PDF)	USN ONR ONR CODE 341 J TANGNEY 875 N RANDOLPH STREET BLDG 87 ARLINGTON VA 22203-1986
1 (PDF)	USA NSRDEC RDNS D D TAMILIO 10 GENERAL GREENE AVE NATICK MA 01760-2642
1 (PDF)	OSD OUSD ATL HPT&B B PETRO 4800 MARK CENTER DRIVE SUITE 17E08 ALEXANDRIA VA 22350

ABERDEEN PROVING GROUND

17 (PDF)	DEVCOM ARL FCDD RLH J LANE Y-S CHEN P FRANASZCZUK K MCDOWELL FCDD RLH F J GASTON K OIE FCDD RLH FA AW EVANS G BOYKIN B FILES FCDD RLH FB J GARCIA (A) H ROY B DALANGIN A OIKNINE FCDD RLH FC J TOURYAN (A) T ROHALY K POLLARD FCDD RLH FD A MARATHE